Opsamling fra sidst?

- Agil skalering
- Geografisk spredning
- Dandomain
- Sitting together – hvorfor *bedst* fysisk
- Taylorism …
- …

Deployment

- "Deploying a single-process monolithic application is a fairly straightforward process"
- Nye teknologier
  - Ikke særligt afprøvede/begrænset erfaring
  - Ikke nødvendigvis kompatible

Fra logisk til fysisk topologi

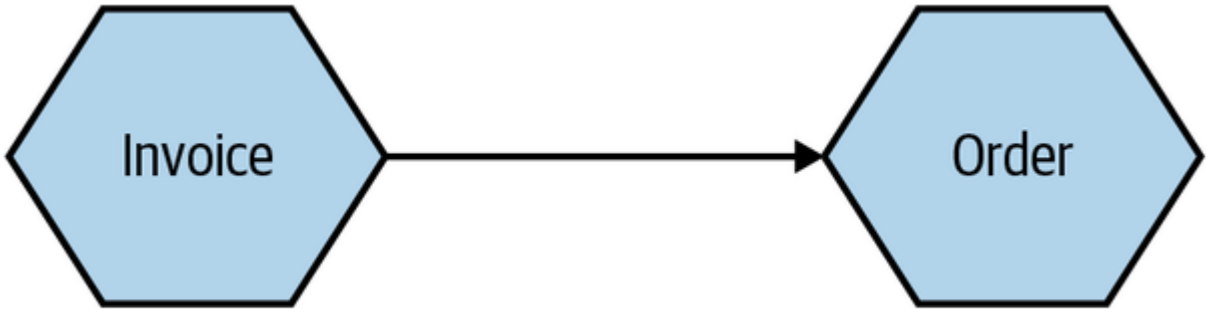


Figure 8-1. A simple, logical view of two microservices

- Hvis vi zoomer ind til den fysiske topologi kan verden se meget mere detaljeret ud.

- Vi skal kunne skifte mellem logisk og fysisk alt efter behov.

Load balancing

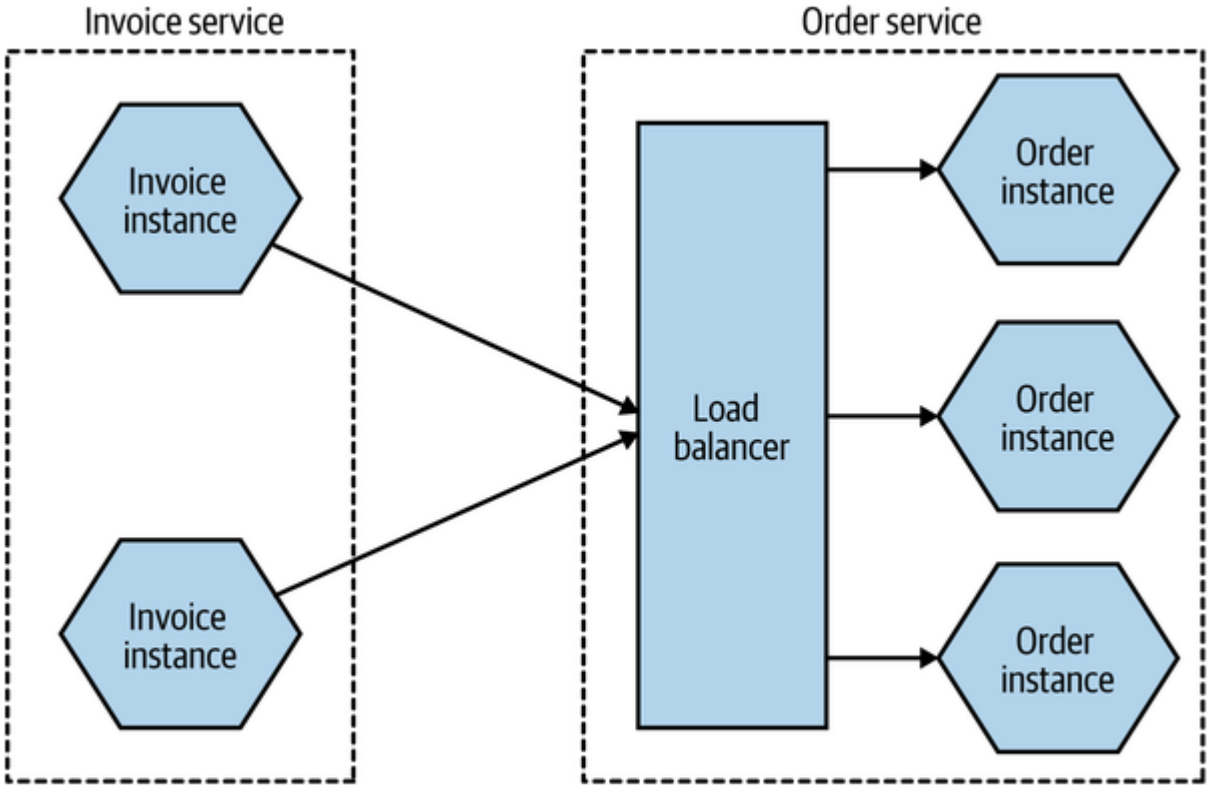


Figure 8-2. Using a load balancer to map requests to specific instances of the `Order` microservice

- Kan klare større belastning
- Større fejltolerance
- “Single point of failure” (oppetid)

Flere datacentre

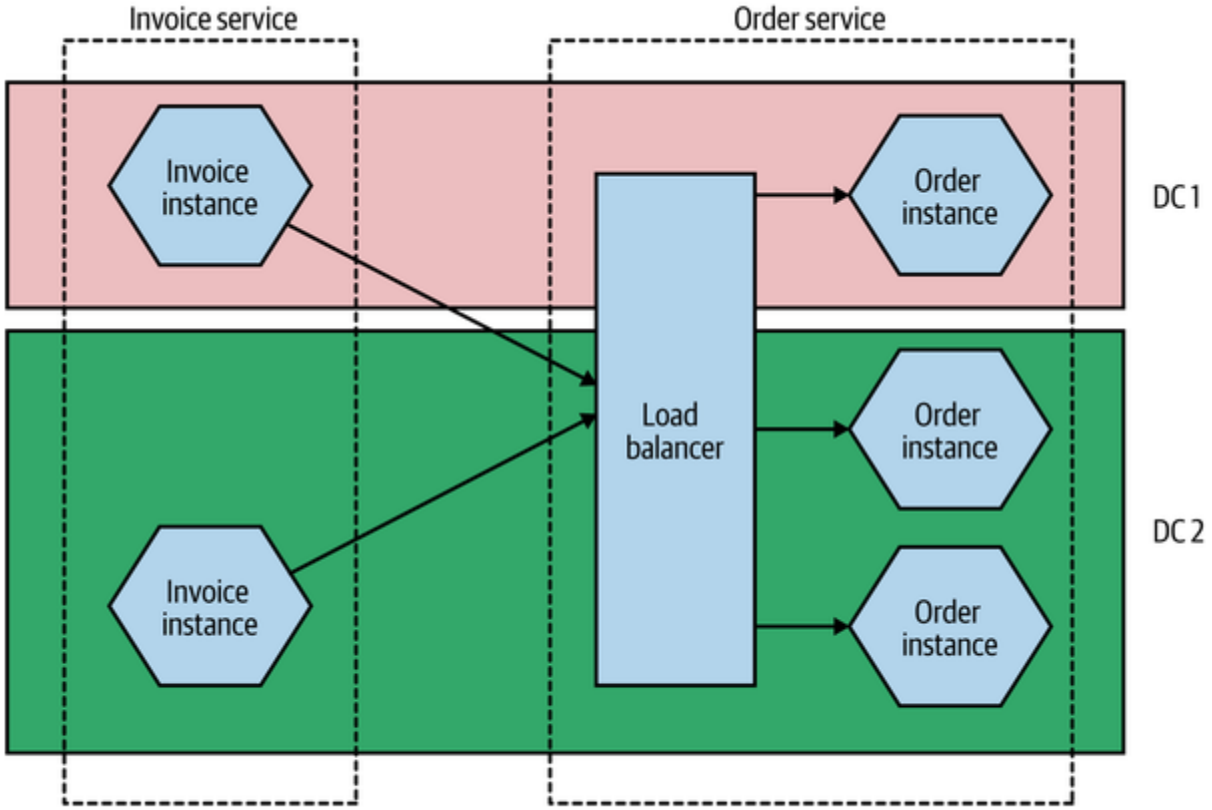


Figure 8-3. Distributing instances across multiple different data centers

- Øget tilgængelighed (FIT/CIA)

Delt database - primærkopi

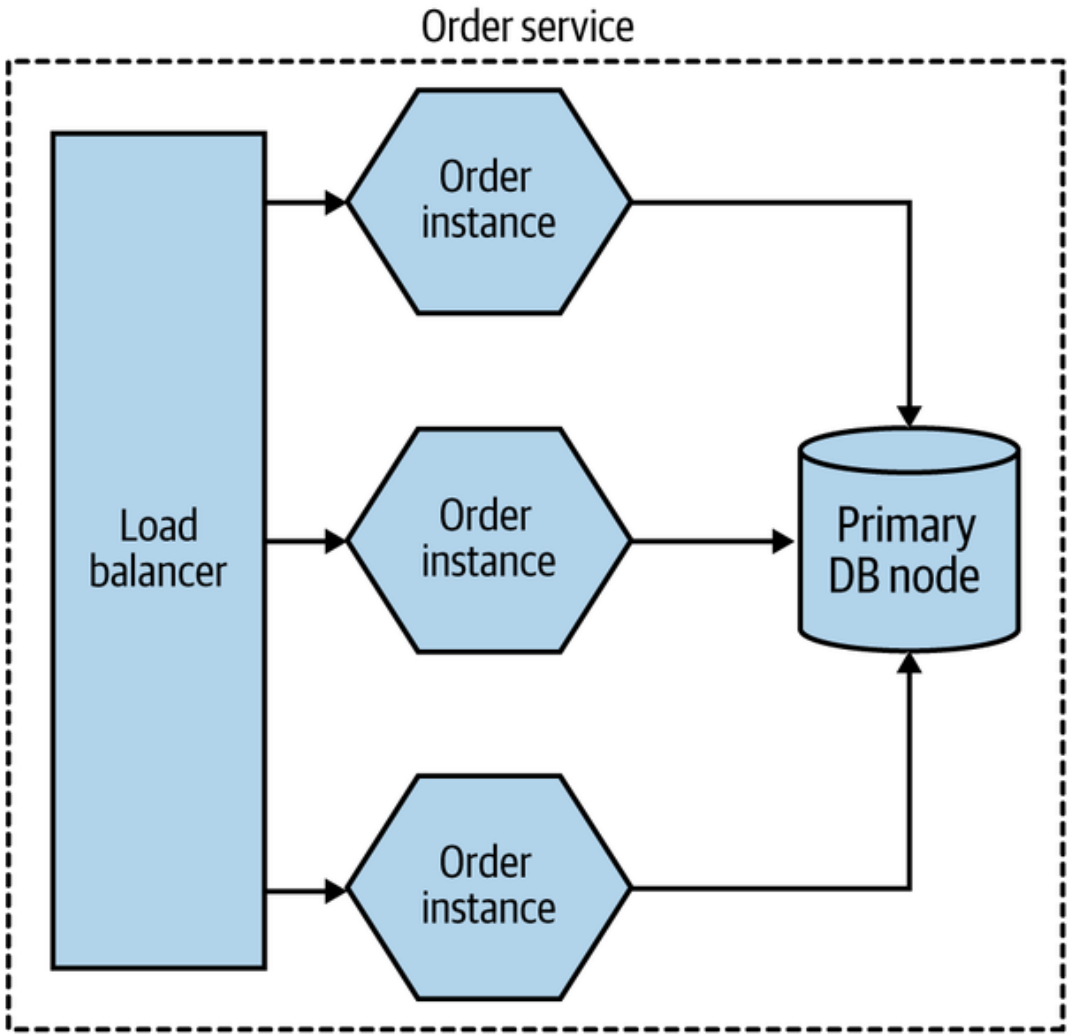


Figure 8-4. Multiple instances of the same microservice can share a database

- Hvorfor?

Replikeret database

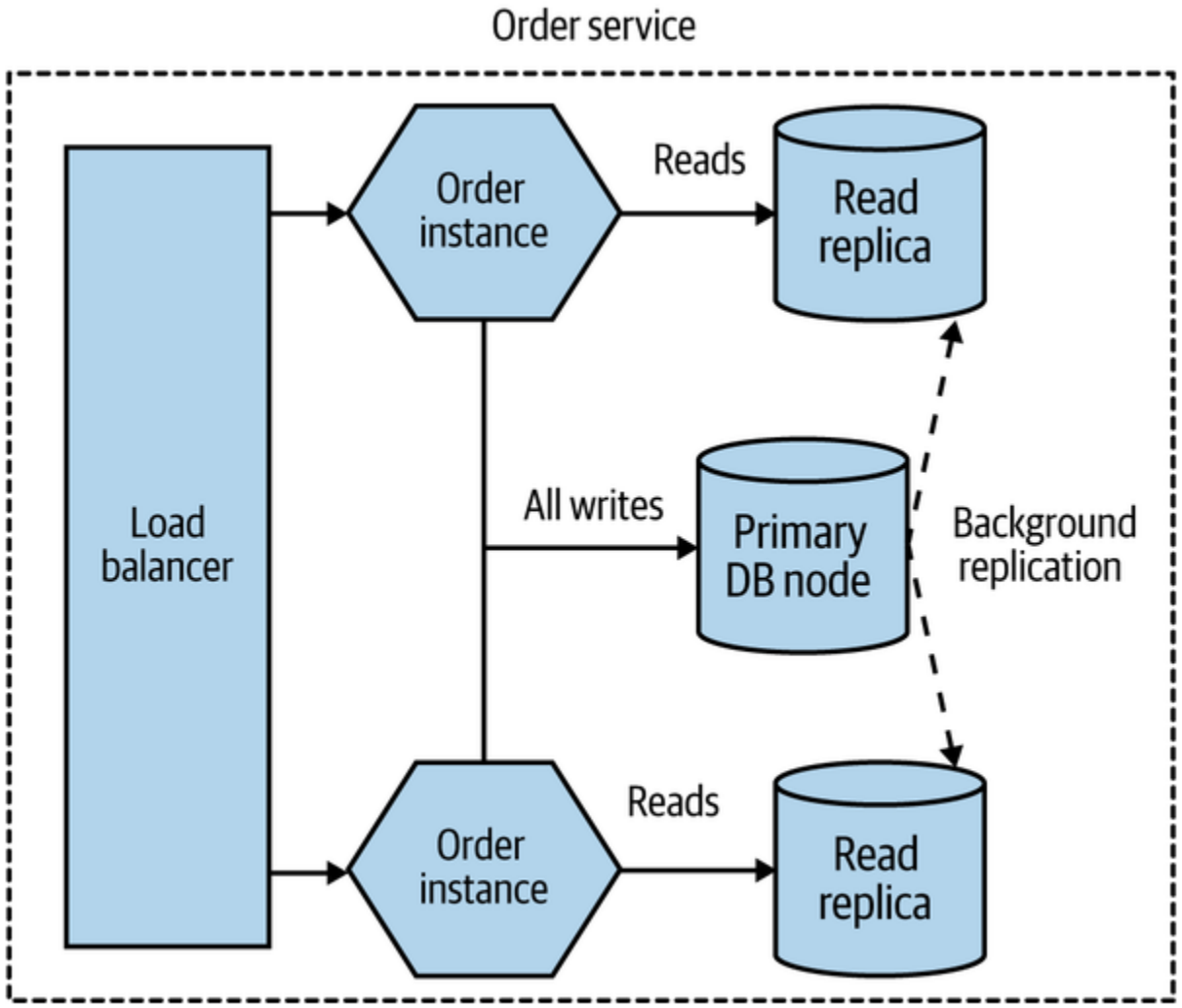


Figure 8-5. Using read replicas to distribute load

- Hvorfor opdeles her i læsning/skrivning

Logisk isoleret DB

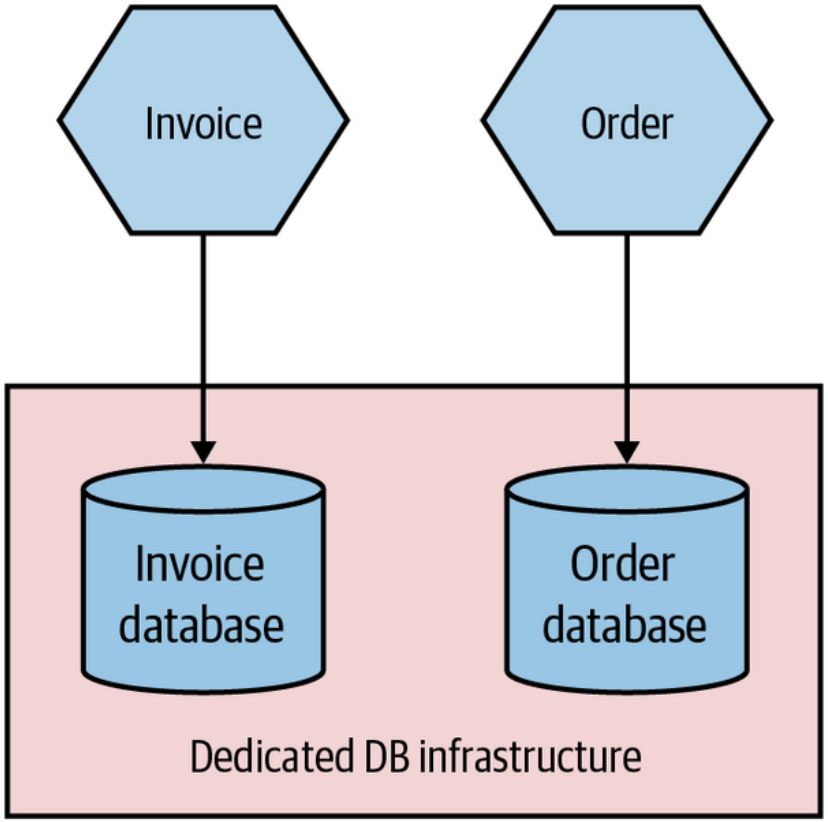


Figure 8-6. The same physical database infrastructure hosting two logically isolated databases

- Sårbarhed?

Fysisk opdelt infrastruktur

ERHVERVSAKADEMI AARHUS

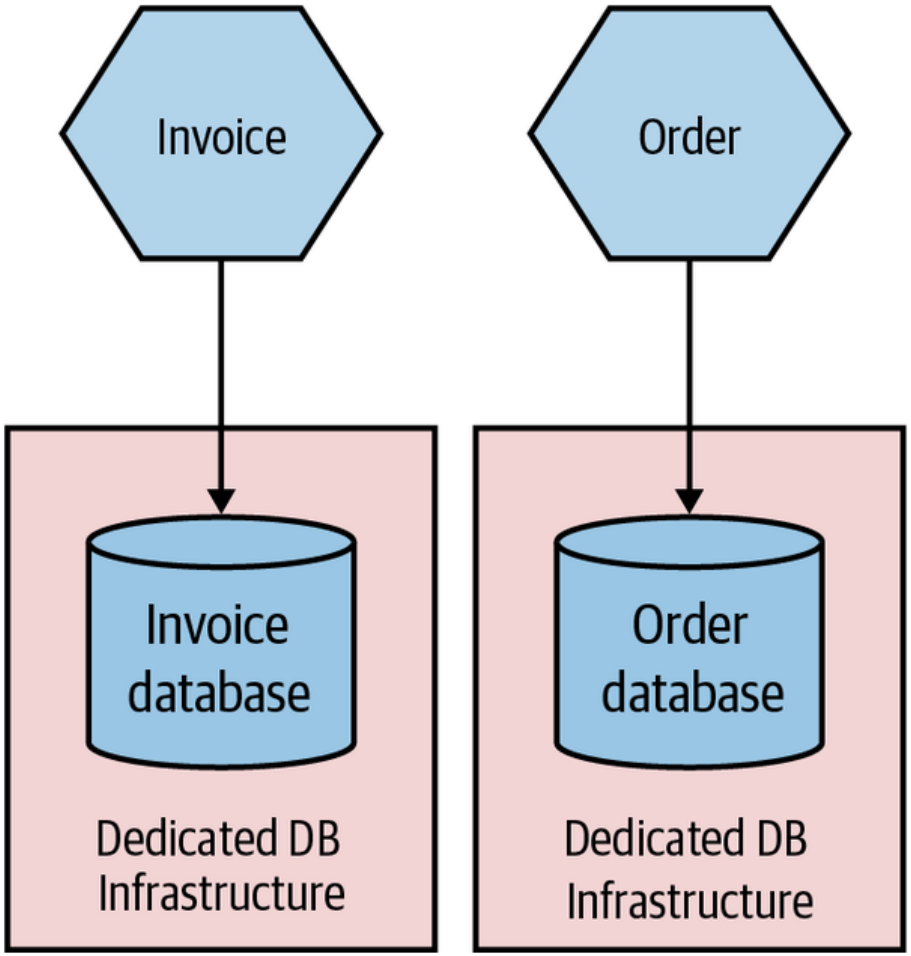


Figure 8-7. Each microservice making use of its own dedicated DB infrastructure

Forskellige omgivelser

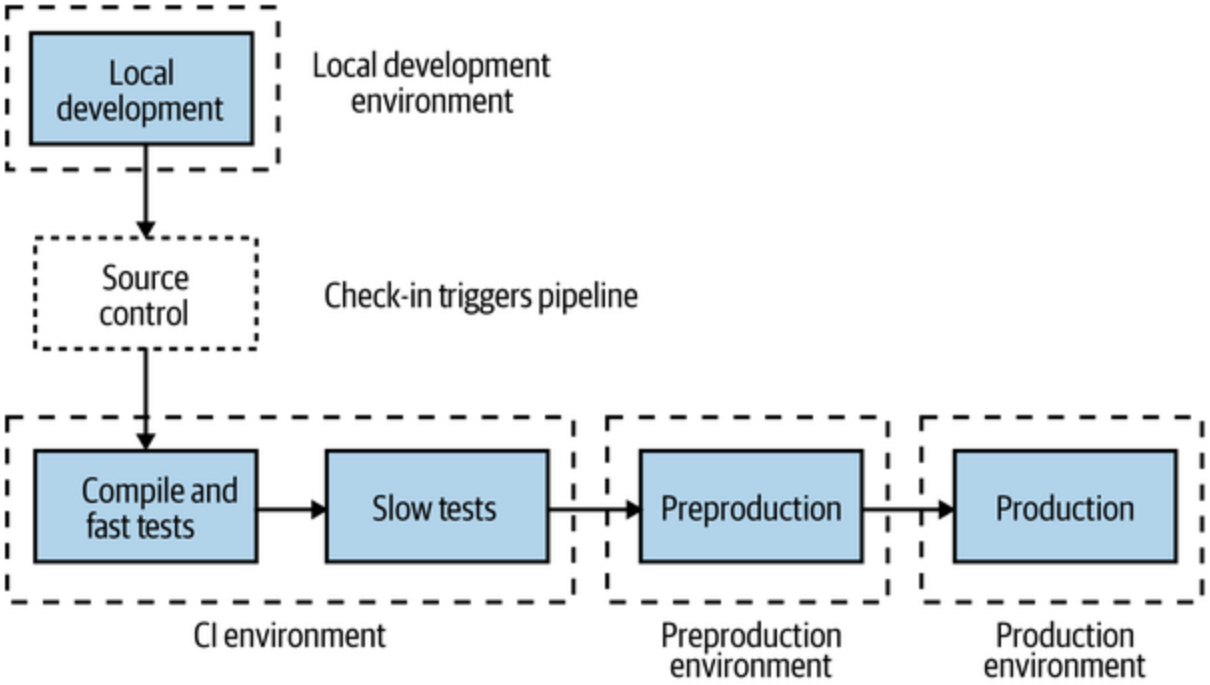


Figure 8-8. Different environments used for different parts of the pipeline

- Forskel på udviklings-, test- og driftsmiljø?
- Hvorfor?

Continuous Integration

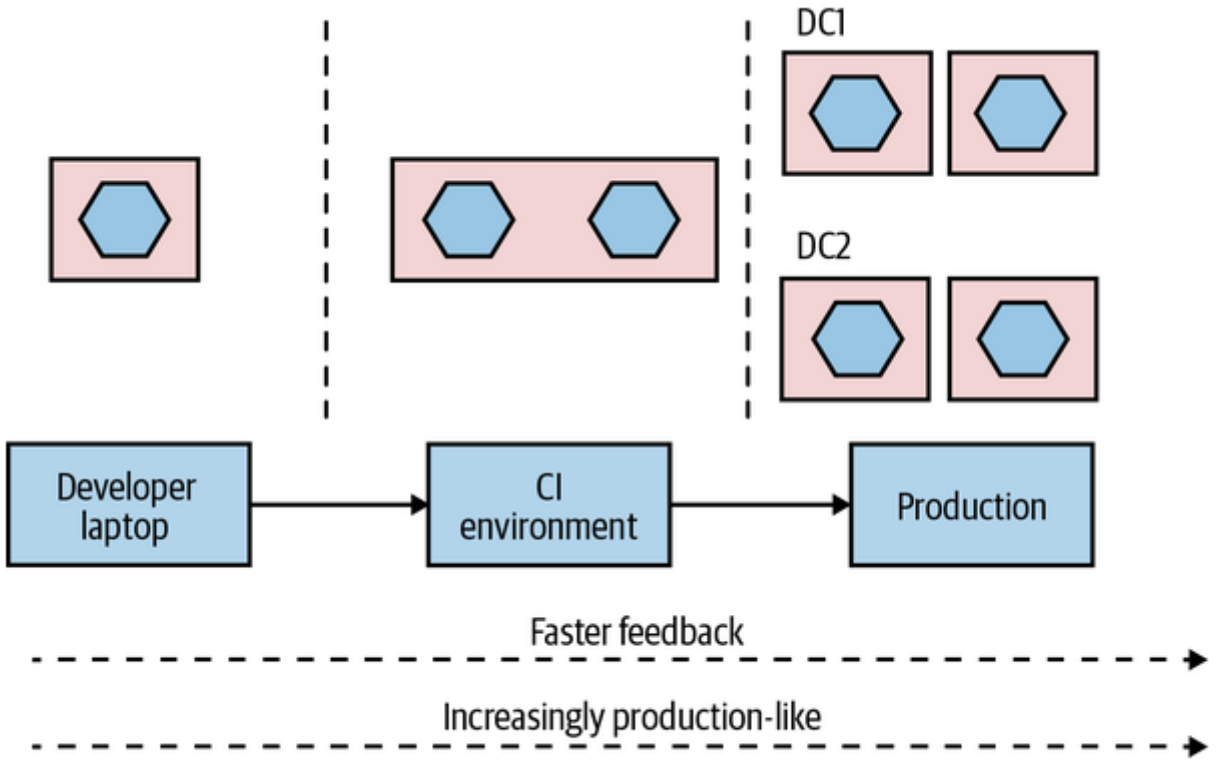


Figure 8-9. A microservice can vary in how it is deployed from one environment to the next

- Hvor er continuous deployment?

Multipel/isoleret udførelse

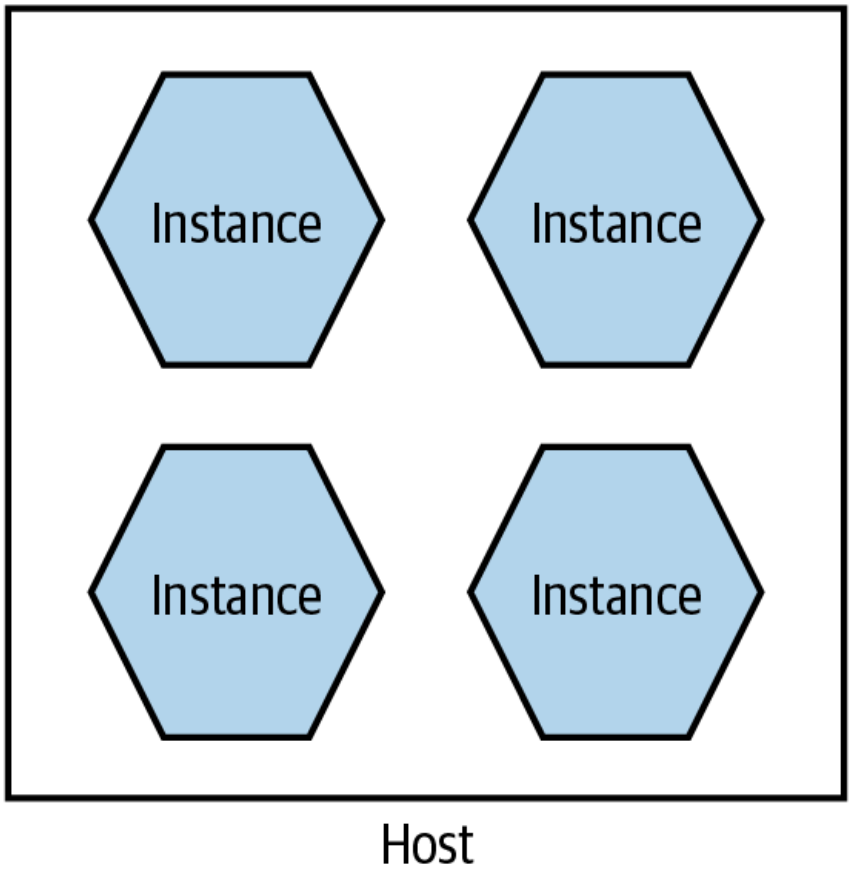


Figure 8-10. Multiple microservices per host

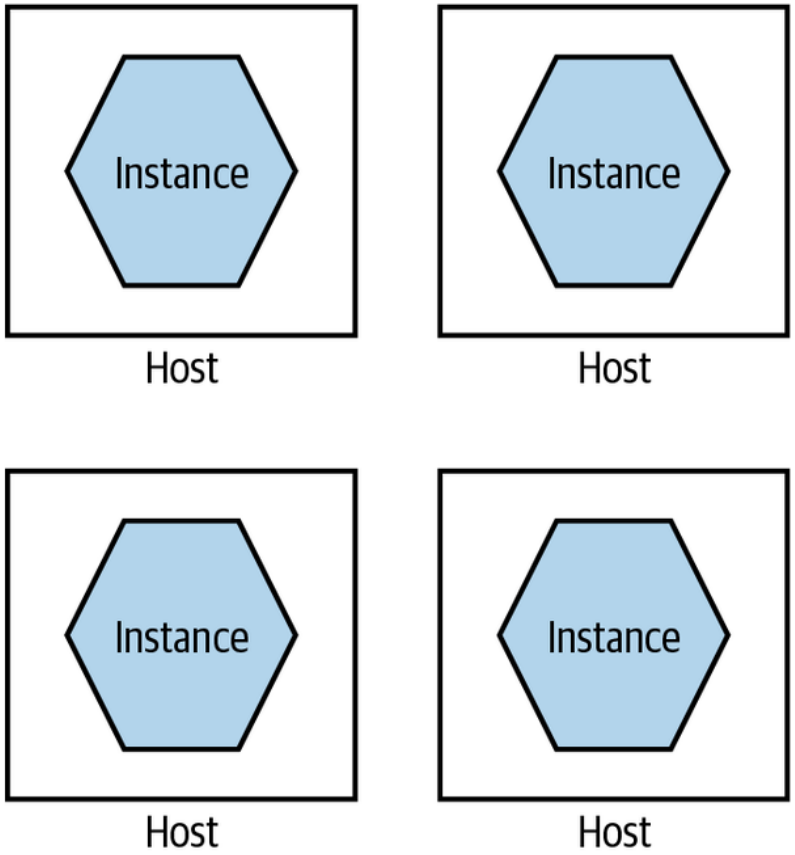


Figure 8-11. A single microservice per host

- Virtualisering?
- Pro et contra?

Trade-offs

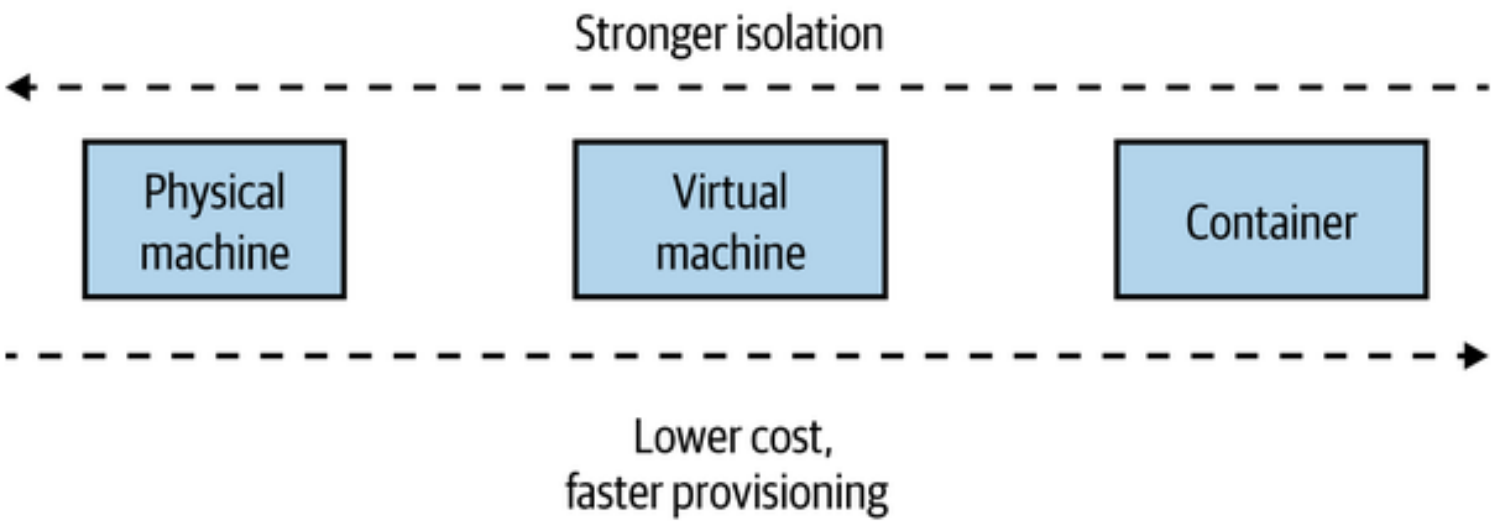


Figure 8-12. Different trade-offs around isolation models

- Selv banker bruger containere … (JN Data)

Management

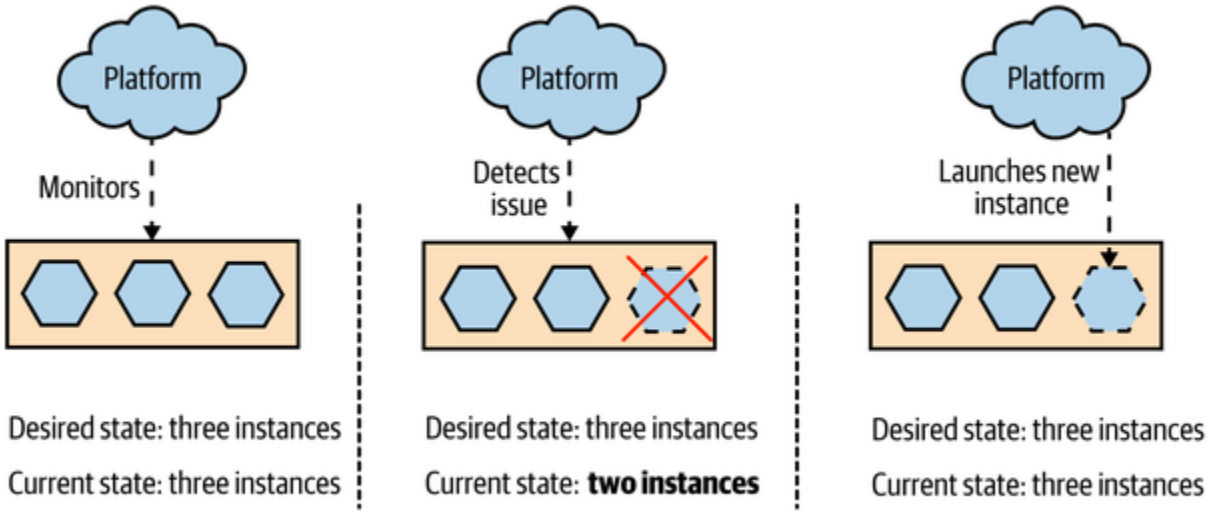


Figure 8-13. A platform providing desired state management, spinning up a new instance when one dies

- Dynamisk management

Opgave 1

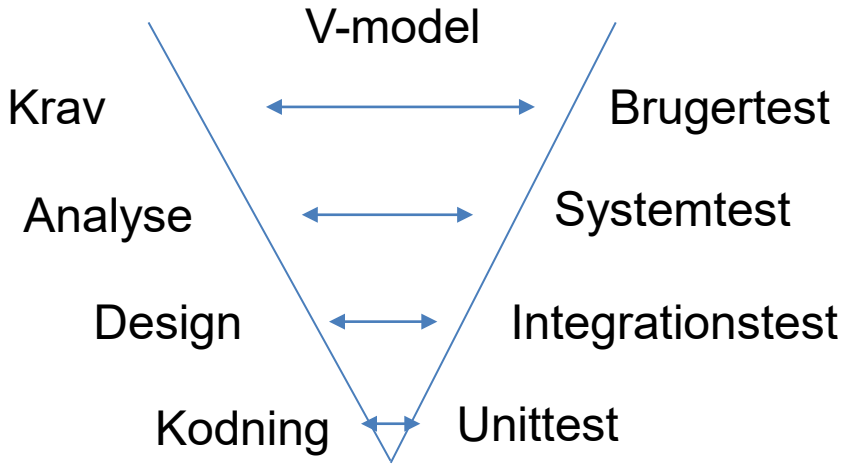
Forklar på skift [Newman] fig. 8.1-13 for hinanden.

Kan nogle af figurerne bruges inden for andre it-emner end microservicer.

Planlægning af test

Testen skal planlægges mht. **hvad**, **hvor meget**, **hvem**, **hvornår**, osv. på hvert af de forskellige **testniveauer,** her de niveauer vi arbejder med

- Unittest
- Integrationstest
- Systemtest
- Brugertest



Unittest

- Test af
  - metoder
  - klasser (som en helhed dvs. attributter, metoder, tilstande osv.)
  - komponenter

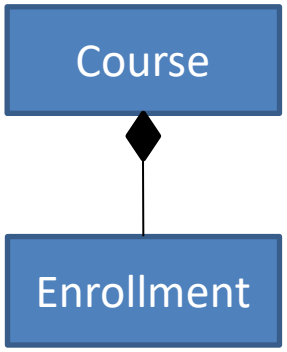
  før de integreres med anden software

```
public int Add(int x, int y)
{
      return x + y;
}
```

Integrationstest

Unittest
Integrationstest
Systemtest
Brugertest

- Tester en gruppe af metoder, klasser og komponenter
  - Test af **sammenhænge mellem klasser**, jvf. klassediagram
  - Test af **sammenhænge mellem andre komponenter**, jvf. arkitektur og evt. packagediagram
  - Test af **brugergrænsefladens funktionalitet**, test af hver funktion
  - Test at **metoder kalder hinanden rigtigt**, at metoder returnerer med det forventede osv.



Systemtest

Unittest
Integrationstest
Systemtest
Brugertest

- Afgør om systemet lever op til de **funktionelle krav** beskrevet i krav og **use cases**

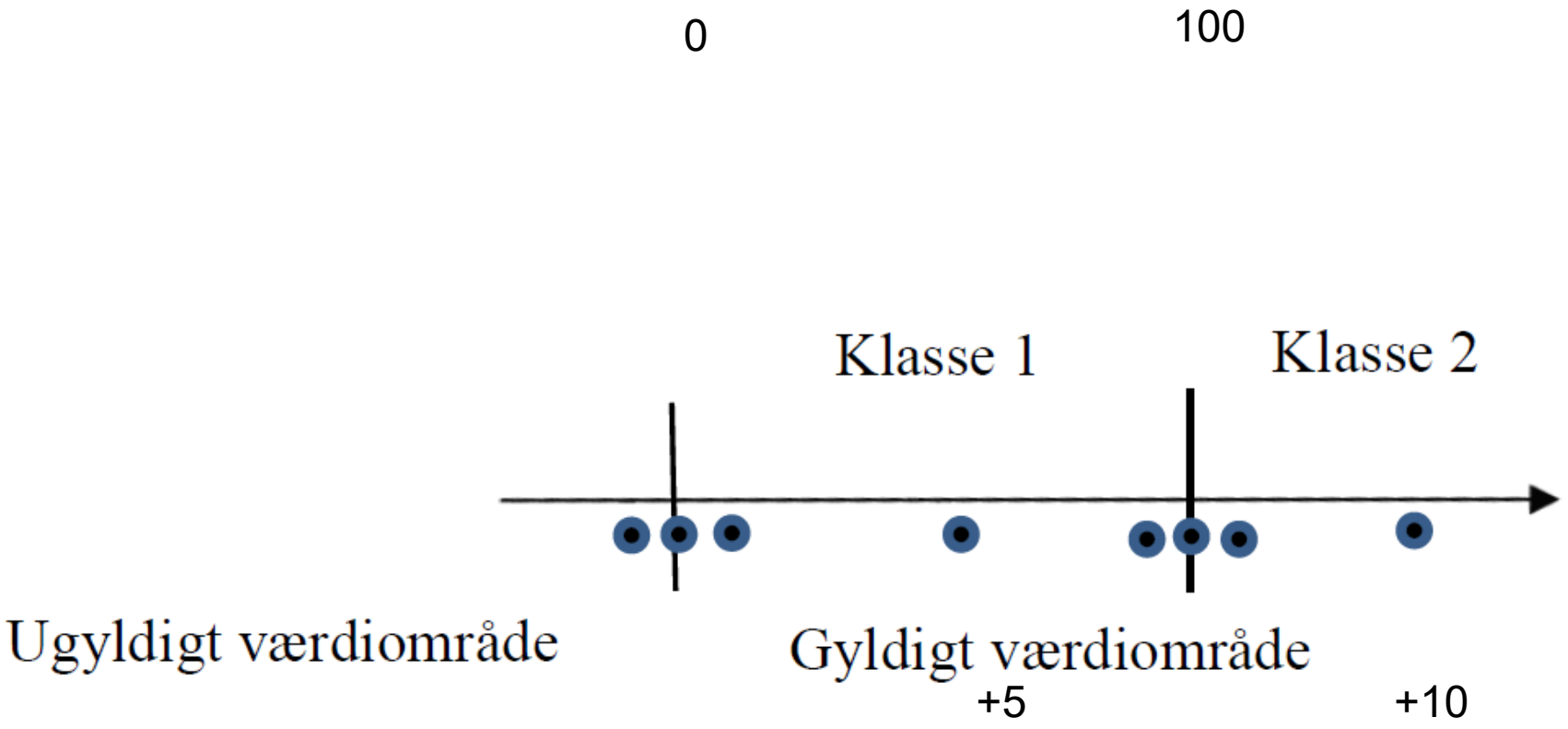
Brugertest

Unittest
Integrationstest
Systemtest
Brugertest

- Test af forskellige scenarier af use casene, om de **fungerer tilfredsstillende** og **tilstrækkeligt brugervenligt** for brugeren

Det kan være test af
- Brugervenlighed, jf. IF-Krav
- Performance, jf. IF-Krav
- Forms/skærmbilleder

Ækvivalensklasser/-mængder



Test

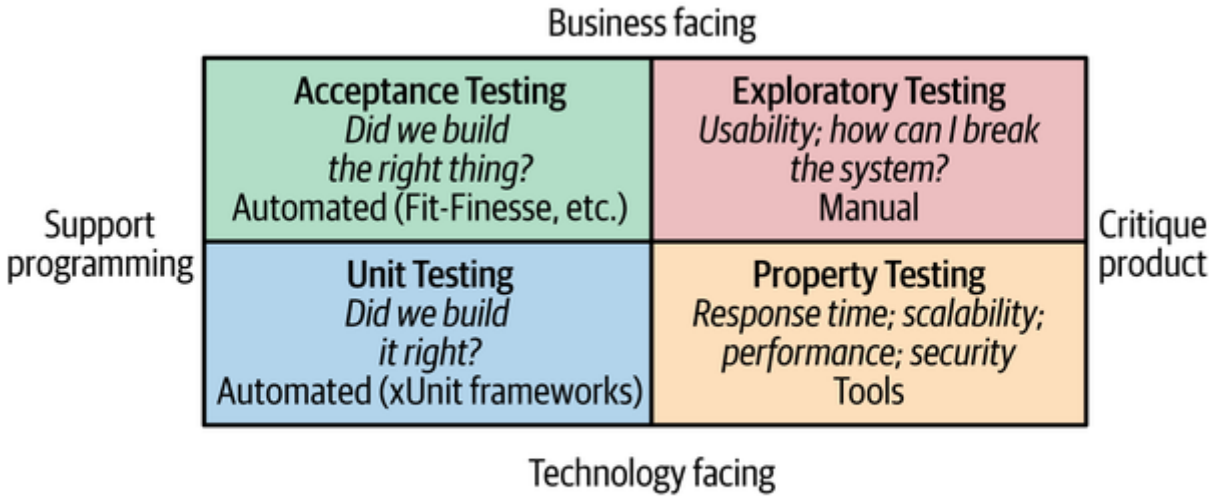


Figure 9-1. Brian Marick's testing quadrant. Lisa Crispin and Janet Gregory, *Agile Testing: A Practical Guide for Testers and Agile Teams*, © 2009

Testpyramide

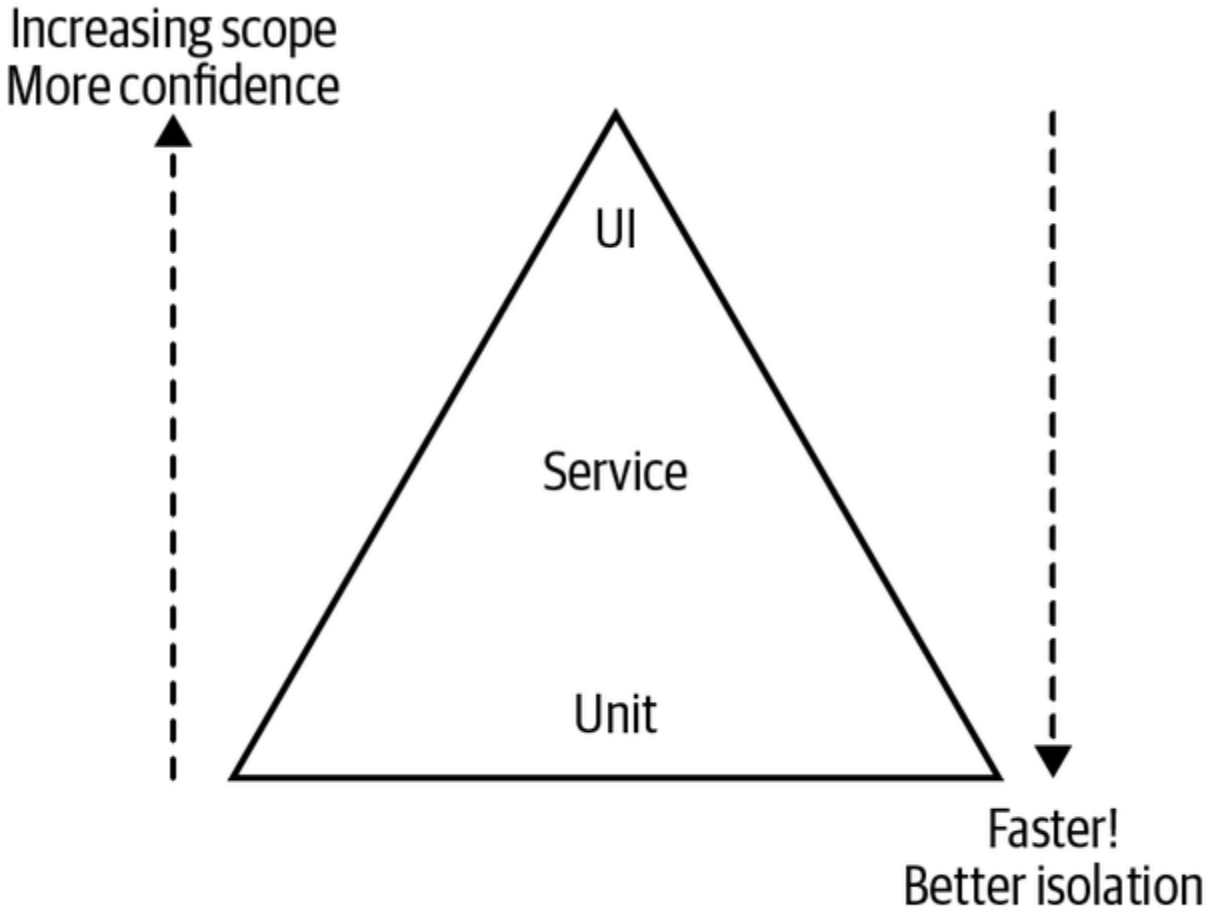


Figure 9-2. Mike Cohn's test pyramid. Mike Cohn, *Succeeding with Agile: Software Development Using Scrum*, 1st ed., © 2010

Test first / TDD

ERHVERVSAKADEMI AARHUS

- Bidrager til at forhindre kode, der svulmer
- Det er lettest at skrive test til kode med lav kobling og høj høj samhørighed
- Stor tillid til kode, der indgår i automatiserede tests
- Rytme. Skriv en test. Indfri testens krav osv.

- Statisk analyse?

- Man skal kunne have tillid til koden!!!
- Gennemsnitlig tid mellem fejl.

- Perfekt versus godt nok?

- Test lavet af programmører og test lavet af kunder?

Test case

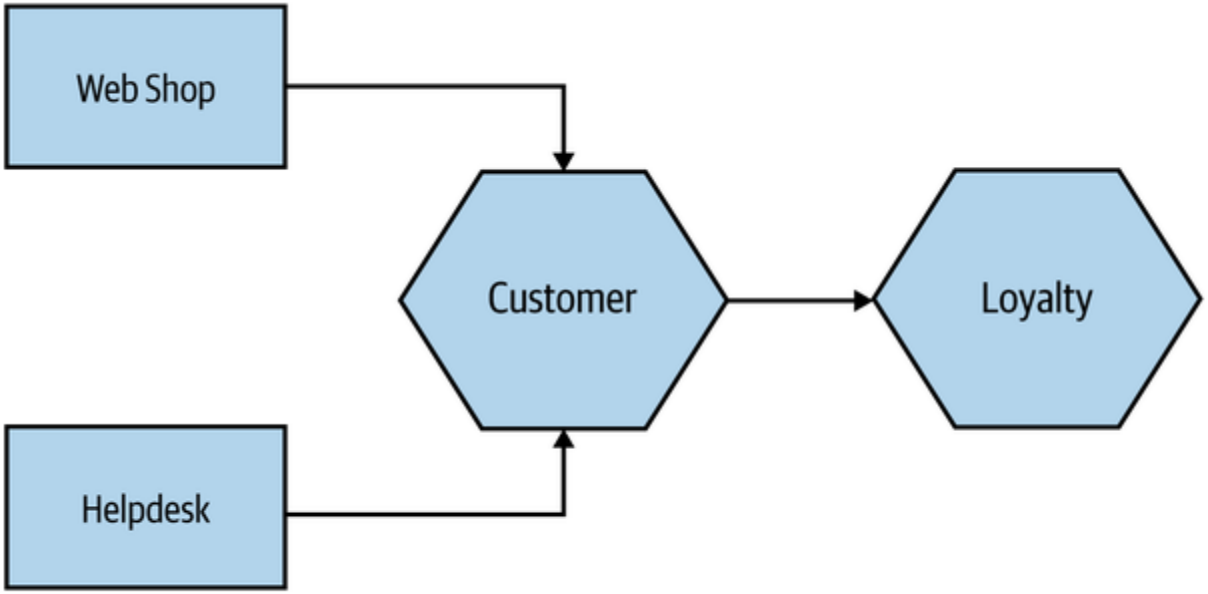


Figure 9-3. Part of our music shop under test

Unit vs. Servicetest

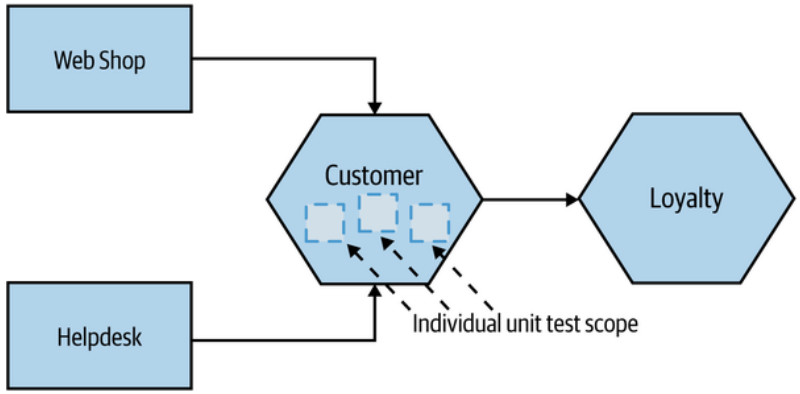


Figure 9-4. Scope of unit tests on our example system

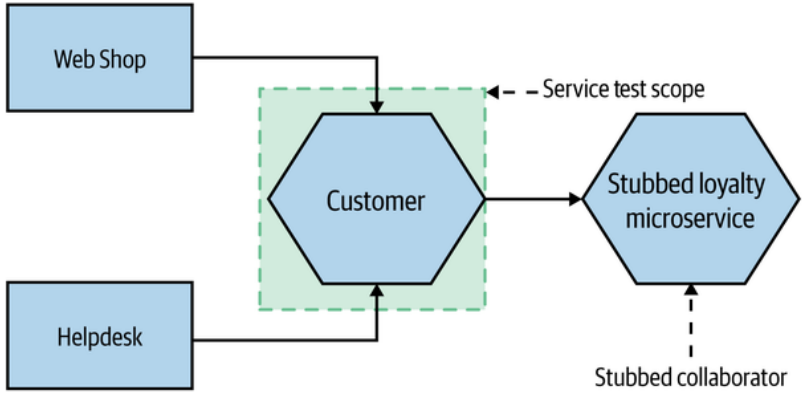


Figure 9-5. Scope of service tests on our example system

- Integrationstest?

End-to-end test

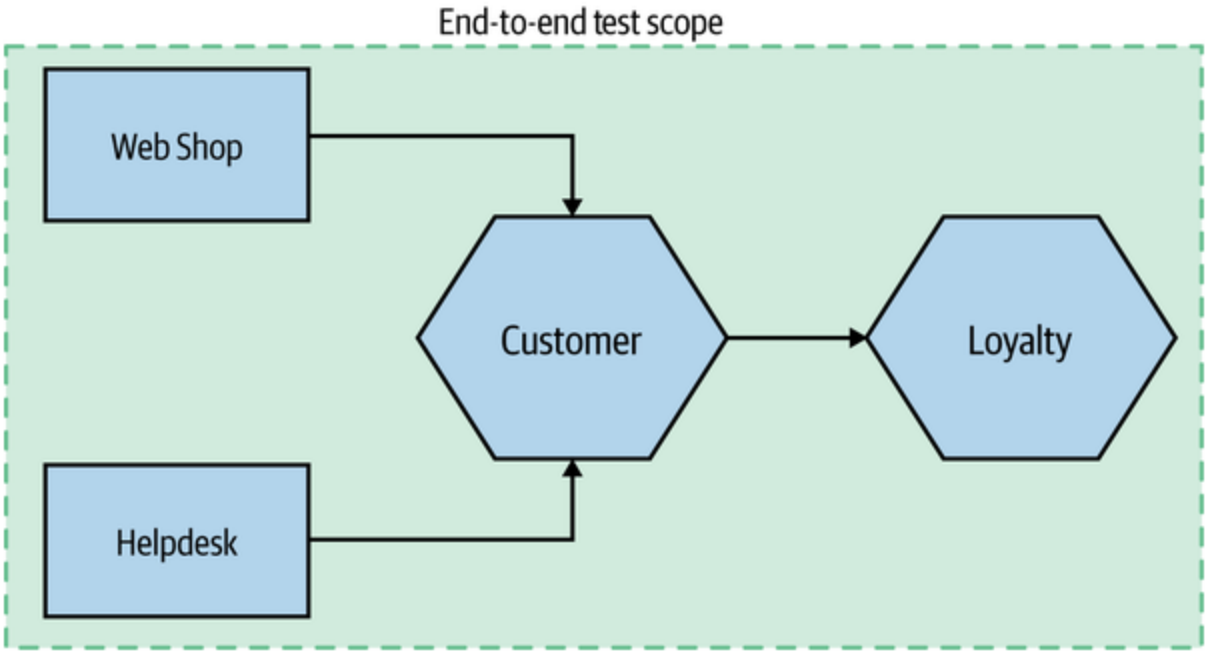


Figure 9-6. Scope of end-to-end tests on our example system

- Integrationstest?

Mock versus Stub

- Stubs svarer det samme ”hver” gang for det samme input.
- Mocks sikrer at kaldet er foretaget. Det vil sige test fejler, hvis kaldet ikke foretages.
- Forskellen er ikke altid lige klar.

https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices:

Fake - A fake is a generic term that can be used to describe either a stub or a mock object. Whether it's a stub or a mock depends on the context in which it's used. So in other words, a fake can be a stub or a mock.
Mock - A mock object is a fake object in the system that decides whether or not a unit test has passed or failed. A mock starts out as a Fake until it's asserted against.
Stub - A stub is a controllable replacement for an existing dependency (or collaborator) in the system. By using a stub, you can test your code without dealing with the dependency directly. By default, a stub starts out as a fake.

End-to-end test

Problem:Test som ”nogle gange” fejler f.eks. pga. netværk, DB eller lignende?

Normalization of deviance

Eradicating Non-Determinism in Tests

Flaky tests …

End-to-end test

Fjern "overflødige" tests – hvis du tør …

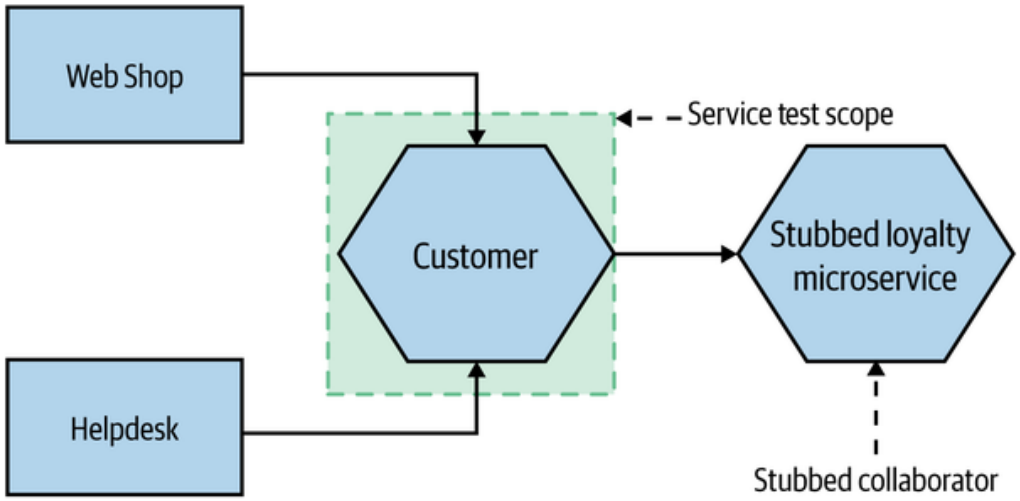


Figure 9-5. Scope of service tests on our example system

Implementering af servicetest

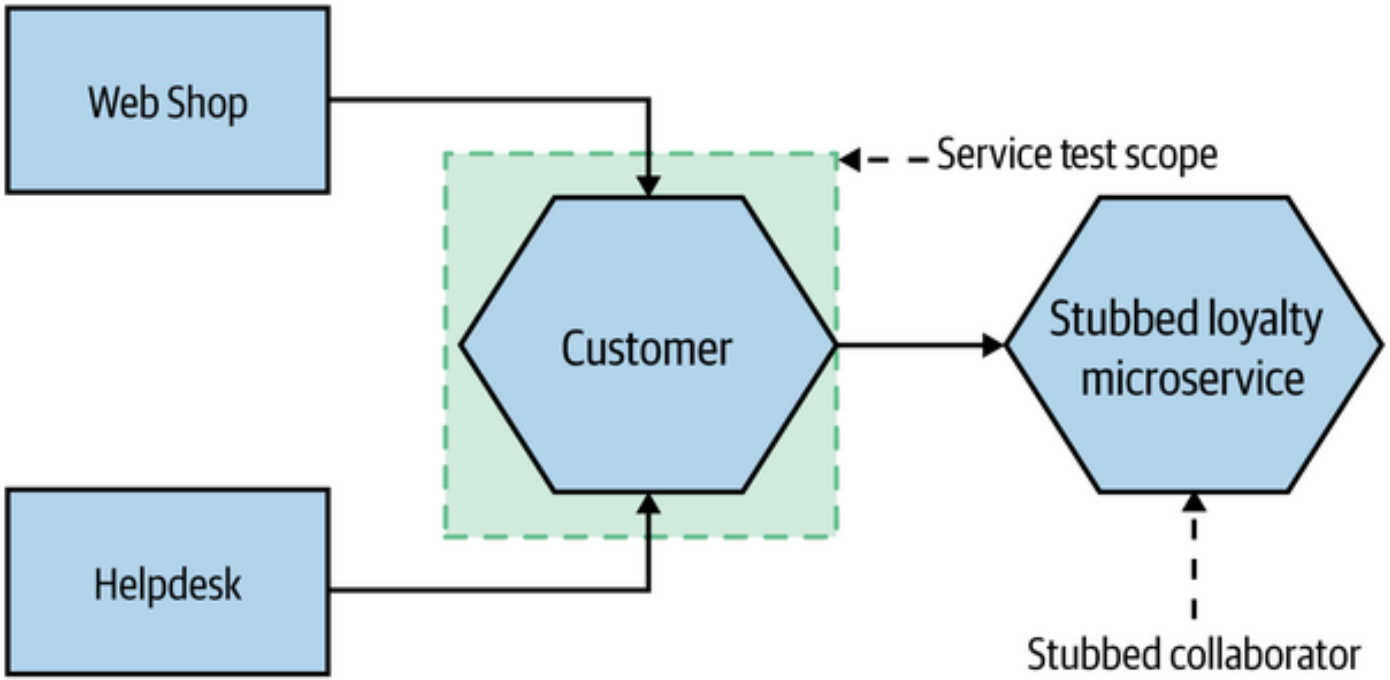


Figure 9-5. Scope of service tests on our example system

Hvis vi ønsker at skrive *service test* for Customer, så ønsker vi at lave stub for Loyalty. Stubs svarer det samme ”hver” gang for det samme input..

End-to-end-test

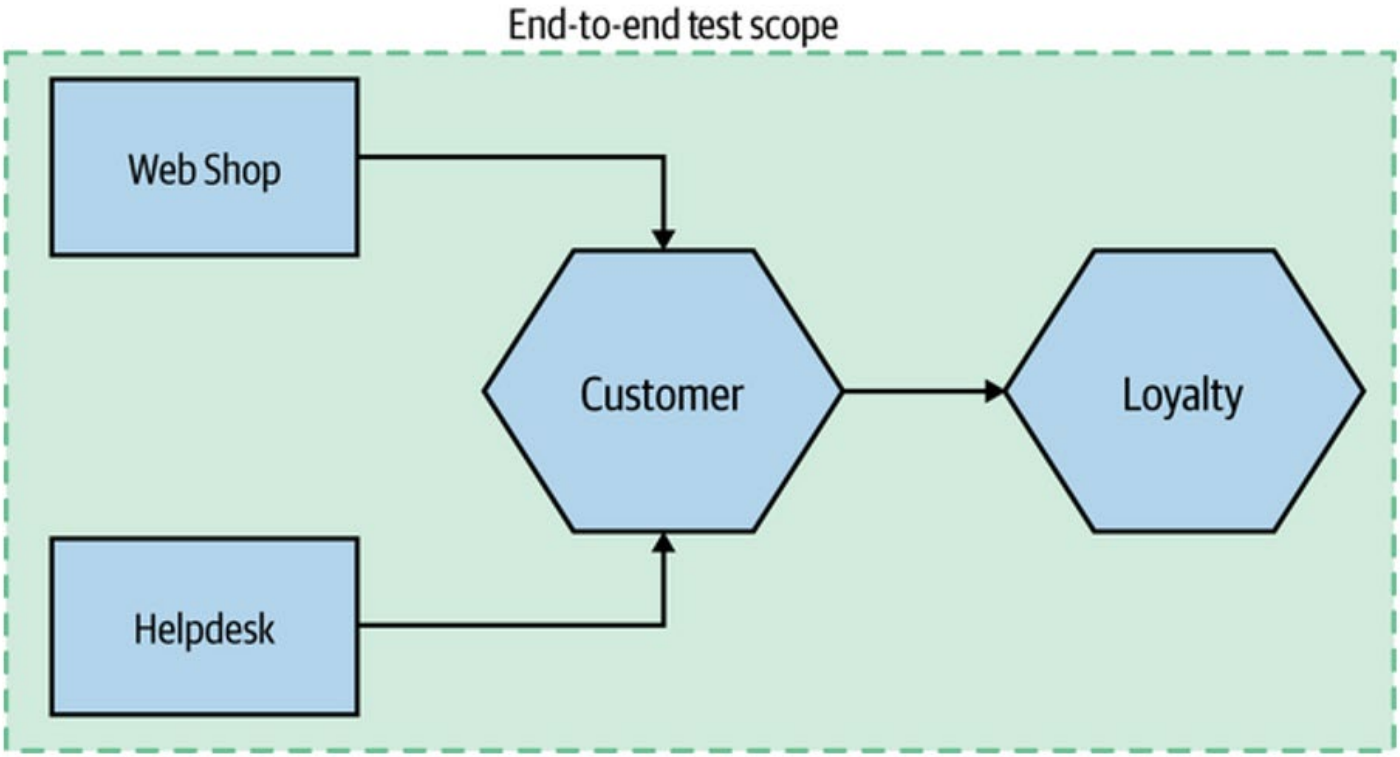


Figure 9-6. Scope of end-to-end tests on our example system

Consumer Driven Contracts (CDC)

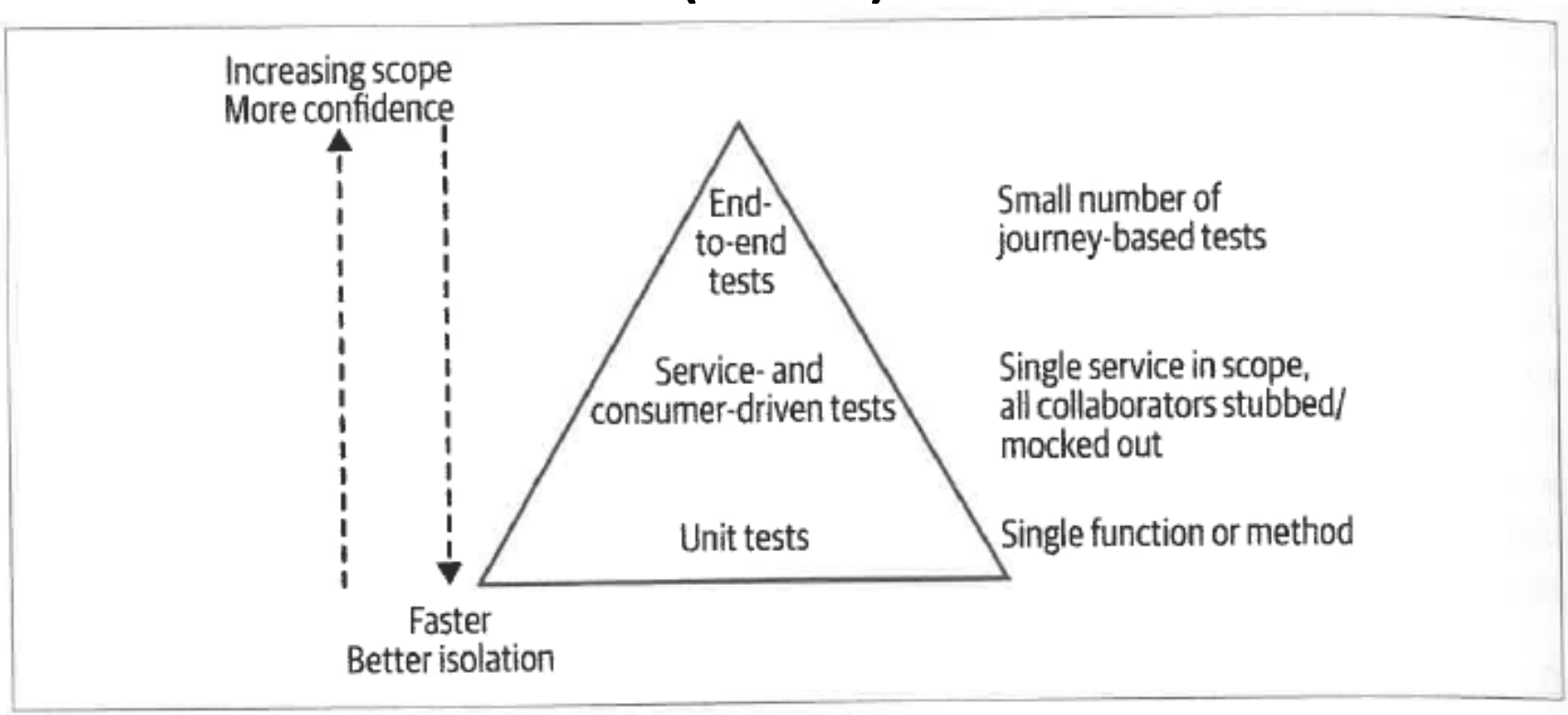


Figure 9-10. Integrating consumer-driven tests into the test pyramid

”Kontraktbaseret udvikling / Design by Contracts”

Opgavearket

ERHVERVSAKADEMI
AARHUS